Naïve Bayes

Let us return yet again to our women athlete example. Suppose I ask you what sport Brittney Griner participates in (gymnastics, marathon running, or basketball) and I tell you she is 6 foot 8 inches and weighs 207 pounds. I imagine you would say basketball and if I ask you how confident you feel about your decision I imagine you would say something along the lines of "pretty darn confident."

Now I ask you what sport Heather Zurich (pictured on the right) plays. She is 6 foot 1 and weighs 176 pounds. Here I am less certain how how will answer. You might say 'basketball' and I ask you how confident you are about your prediction. You probably are less confident than you were about your prediction for Brittney Griner. She could be a tall marathon runner.

Finally, I ask you about what sport Yumiko Hara participates in; she is 5 foot 4 inches tall and weighs 95 pounds. Let's say you say 'gymnastics' and I ask how confident you feel about your decision. You will probably say something along the lines of "not too confident." A number of marathon runner have similar heights and weights.

With the nearest neighbor algorithms, it is difficult to quantify confidence about a classification. With classification methods based on probability—

Bayesian methods—we can not only make a classification but we can make probabilistic classifications—this athlete is 80% likely to be a basketball player, this patient has a 40% chance of getting diabetes in the next five years, the probability of rain in Las Cruces in the next 24 hours is 10%.

Nearest Neighbor approaches are called **lazy learners**. They are called this because when we give them a set of training data, they just basically save—or remember—the set. Each time it classifies an instance, it goes through the entire training dataset. If we have a 100,000 music tracks in our training data, it goes through the entire 100,000 tracks each time it classifies an instance.

Bayesian methods are called **eager learners**. When given a training set eager learners immediately analyze the data and build a model. When it wants to classify an instance it uses this internal model. Eager learners tend to classify instances faster than lazy learners.

The ability to make probabilistic classifications, and the fact that they are eager learners are two advantages of Bayesian methods.

Probability

I am assuming you have some basic knowledge of probability. I flip a coin; what is the probably of it beings a 'heads'? I roll a 6 sided fair die, what is the probability that I roll a '1'? that sort of thing. I tell you I picked a random 19 year old and have you tell me the probability of that person being female and without doing any research you say 50%. These are example of what is called prior probability and is denoted $P(h)$—the probability of hypothesis h.

So for a coin:

P(heads) = 0.5

For a six sided dice, the probability of rolling a '1':

P(1) = 1/6

If I have an equal number of 19 yr. old male and females →

P(female) = .5

Suppose I give you some additional information about that 19 yr. old—the person is a student at the Frank Lloyd Wright School of Architecture in Arizona. You do a quick Google search, see that the student body is 86% female and revise your estimate of the likelihood of the person being female to 86%.

This we denote as $P(h|D)$ —the probability of the hypothesis h given some data D. For example:

P(female | attends Frank Lloyd Wright School) = 0.86

which we could read as "The probability the person is female given that person attends the Frank Lloyd Wright School is 0.86

The formula is

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

An example.

In the following table I list some people and the types of laptops and phones they have:

name	laptop	phone
Kate	PC	Android
Tom	PC	Android
Harry	PC	Android
Annika	Mac	iPhone
Naomi	Mac	Android
Joe	Mac	iPhone
Chakotay	Mac	iPhone
Neelix	Mac	Android
Kes	PC	iPhone
B'Elanna	Mac	iPhone

What is the probability that a randomly selected person uses an iPhone?

There are 5 iPhone users out of 10 total users so

$$P(iPhone) = \frac{5}{10} = 0.5$$

What is the probability that a randomly selected person uses an iPhone given that person uses a Mac laptop?

$$P(iPhone \mid mac) = \frac{P(mac \cap iPhone)}{P(mac)}$$

First, there are 4 people who use both a Mac and an iPhone:

$$P(mac \cap iPhone) = \frac{4}{10} = 0.4$$

and the probability of a random person using a mac is

$$P(mac) = \frac{6}{10} = 0.6$$

So the probability of that some person uses an iPhone given that person uses a Mac is

$$P(iPhone \mid mac) = \frac{0.4}{0.6} = 0.667$$

That is the formal definition of posterior probability. Sometimes when we implement this we just use raw counts:

$$P(iPhone|mac) = \frac{number\ of\ people\ who\ use\ a\ mac\ and\ an\ iPhone}{number\ of\ people\ who\ use\ a\ mac}$$

$$P(iPhone \mid mac) = \frac{4}{6} = 0.667$$

sharpen your pencil

What's the probability of a person owning a mac given that they own an iPhone

i.e., P(mac|iPhone)?

tip

If you feel you need practice with basic probabilities please see the links to tutorials at guidetodatamining.com.

sharpen your pencil – solution

What's the probability of a person owning a mac given that they own an iphone

i.e., P(mac|iPhone)?

$$P(mac \mid iPhone) = \frac{P(iPhone \cap mac)}{P(iPhone)}$$

$$= \frac{0.4}{0.5} = 0.8$$

Some terms:

$P(h)$, the probability that some hypothesis h is true, is called the **prior probability** of h. Before we have any evidence, the probability of a person owning a Mac is 0.6 (the evidence might be knowing that the person also owns an iPhone).

P(h|d) is called the **posterior probability** of h. After we observe some data d what is the probability of h? For example, after we observe that a person owns an iPhone, what is the probability of that same person owning a Mac? It is also called **conditional probability**.

In our quest to build a Bayesian Classifier we will need two additional probabilities, $P(D)$ and $P(D|h)$. To explain these consider the following example.

Microsoft Shopping Cart

Did you know that Microsoft makes smart grocery store shopping carts? Yep, they do. Well, actually, Microsoft has contracted with a company called Chaotic Moon to develop them. Chaotic Moon's slogan is *We are smarter than you. We are more creative than you.* You can decide whether they are arrogant, cheeky, or something else. Anyway, the cart combines a shopping cart with a Windows 8 tablet, a Kinect, a Bluetooth speaker (so the cart can talk to you), and a mobile robotics platform (so the cart can follow you around the store).

You come in with your grocery store loyalty card. The cart recognizes you. It has recorded all previous purchases (as well as the purchases of everyone else in the store).

Suppose the cart software wants to determine whether to show you a targeted ad for Japanese Sensha Green Tea. It only wants to show that ad if you are likely to purchase the tea.

The cart system has accumulated the small dataset shown on the next page from other shoppers

P(D) is the probability that some training data will be observed. For example, looking on the next page we see that the probability that the zip code will be 88005 is 5/10 or 0.5.

P(88005) = 0.5

P(D|h) is the probability that some data value holds given the hypothesis. For example, the probability of the zip code being 88005 given that the person bough Sencha Green Tea or *P*(*88005*|*Sencha Tea*).

Customer ID	Zipcode	bought organic produce?	bought Sencha green tea?
1	88005	Yes	Yes
2	88001	No	No
3	88001	Yes	Yes
4	88005	No	No
5	88003	Yes	No
6	88005	No	Yes
7	88005	No	No
8	88001	No	No
9	88005	Yes	Yes
10	88003	Yes	Yes

Zipcodes are a set of postal codes used in the U.S.

In this case we are looking at all the instances where the person bought Sensha Tea. There are 5 such instances. Of those, 3 are with the 88005 zip code.

$$P(88005 \mid SenchaTea) = \frac{3}{5} = 0.6$$

sharpen your pencil

What's the probability of the zip code being 88005 given that the person did not buy Sencha tea?

sharpen your pencil – solution

What's the probability of the zip code being 88005 given that the person did not buy Sencha tea?

There are 5 occurrences of a person not buying Sencha tea. Of those, 2 lived in the 88005 zip code. So

$$P(88005 \mid \neg SenchaTea) = \frac{2}{5} = 0.4$$

That ¬ symbol means 'not'.

sharpen your pencil

This is key to understanding the rest of the chapter so let us practice just a bit more.

1. What is the probability of a person being in the 88001 zipcode (without knowing anything else)?

2. What is the probability of a person being in the 88001 zipcode knowing that they bought Sencha tea?

3. What is the probability of a person being in the 88001 zipcode knowing that they did not buy Sencha tea?

sharpen your pencil – solution

This is key to understanding the rest of the chapter so let us practice just a bit more.

1. What is the probability of a person being in the 88001 zipcode (without knowing anything else)?

   There are 10 total entries in our database and only 3 of them are from 88001 so P(88001) is 0.3

2. What is the probability of a person being in the 88001 zipcode knowing that they bought Sencha tea?
   There are 5 instances of buying Sencha tea and only 1 of them is from the 88001 zipcode so

$$P(88001 \mid SenchaTea) = \frac{1}{5} = 0.2$$

3. What is the probability of a person being in the 88001 zipcode knowing that they did not buy Sencha tea?
   There are 5 instances of not buying Sencha tea and 2 of them are from the 88001 zipcode:

$$P(88001 \mid \neg SenchaTea) = \frac{2}{5} = 0.4$$

Bayes Theorem

Bayes Theorem describes the relationship between $P(h)$, $P(h|D)$, $P(D)$, and $P(D|h)$:

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

This theorem is the cornerstone of all Bayesian methods. Usually in data mining we use this theorem to decide among alternative hypotheses. Given the evidence, is the person a gymnast, marathoner, or basketball player. Given the evidence, will this person buy Sencha tea, or not. To decide among alternatives we compute the probability for each hypothesis. For example,

We want to display an ad for Sencha Tea on our smart shopping cart display only if we think that person is likely to buy the tea. We know that person lives in the 88005 zipcode.

There are two competing hypotheses:

The person will buy Sencha tea.
We compute P(buySenchaTea|88005)

The person will not buy Sencha tea.
We compute P(¬buySenchaTea|88005)

We pick the hypothesis with the highest probability!

So if P(buySenchaTea|88005) = 0.6 and

P(¬buySenchaTea|88005) = 0.4

So it is more likely that the person will buy the tea so we will display the ad.

Suppose we work for an electronics store and we have three sales flyers in email form. One flyer features a laptop, another features a desktop and the final flyer a tablet. Based on what we know about each customer we will email that customer the flyer that will most likely generate a sale. For example, I may know that a customer lives in the 88005 zipcode, that she has a college age daughter living at home, and that she goes to yoga class. Should I send her the flyer with the laptop, desktop, or tablet?

Let D represent all that I know about that customer:

- lives in 88005 zipcode
- has college age daughter
- goes to yoga class

My hypotheses are which flyer is the best: laptop, desktop, tablet. So I compute:

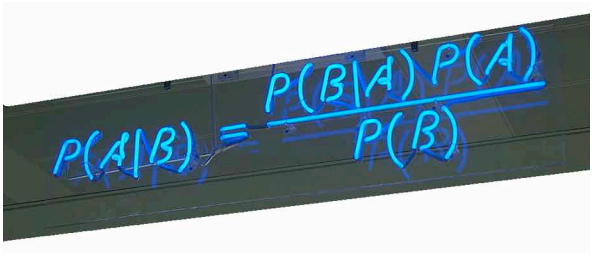
$$P(laptop \mid D) = \frac{P(D \mid laptop)P(laptop)}{P(D)}$$

$$P(desktop \mid D) = \frac{P(D \mid desktop)P(desktop)}{P(D)}$$

$$P(tablet \mid D) = \frac{P(D \mid tablet)P(tablet)}{P(D)}$$

And pick the hypothesis with the highest probability.

More abstractly, in a classification task we have a number of possible hypotheses: h_1, h_2, ...h_n. These hypotheses are the different categories of our task (for example, basketball players, marathoners, gymnasts, or 'will get diabetes', 'will not get diabetes').



$$P(h_1 \mid D) = \frac{P(D \mid h_1)P(h_1)}{P(D)} \qquad P(h_2 \mid D) = \frac{P(D \mid h_2)P(h_2)}{P(D)}$$

,

$$\ldots \qquad P(h_n \mid D) = \frac{P(D \mid h_n)P(h_n)}{P(D)}$$

Once we compute all these probabilities, we will pick the hypothesis with the highest probability. This is called **the maximum a posteriori hypothesis**, or h_{MAP}.

We can translate that English description of calculating the maximum a posteriori hypothesis into the following formula:

$$h_{MAP} = \arg\max_{h \in H} P(h \mid D)$$

H is the set of all the hypotheses. So $h \in H$ means "for every hypothesis in the set of hypotheses." The full formula means something like "for every hypothesis in the set of hypotheses compute $P(h|D)$ and pick the hypothesis with the largest probability." Using Bayes Theorem we can convert that formula to:

$$h_{MAP} = \arg\max_{h \in H} \frac{P(D \mid h)P(h)}{P(D)}$$

So for every hypothesis we are going to compute:

$$\frac{P(D \mid h)P(h)}{P(D)}$$

You might notice that for all these calculations, the denominators are identical—$P(D)$. Thus, they are independent of the hypotheses. If a specific hypothesis has the max probability with the formula used above, it will still be the largest if we did not divide all the hypotheses by $P(D)$. If our goal is to find the most likely hypothesis, we can simplify our calculations:

$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

To see how this works, we will use an example from Tom M. Mitchell's book, *Machine Learning*. Tom Mitchell is chair of the Machine Learning Department at Carnegie Mellon University. He is a great researcher and an extremely nice guy. On to the example from the book. Consider a medical domain where we want to determine whether a patient has a particular kind of cancer or not. We know that only 0.8% of the people in the U.S. have this form of cancer. There is a simple blood test we can do that will help us determine whether someone has it. The test is a binary one—it comes back either POS or NEG. When the disease is present the test returns a correct POS result 98% of the time; it returns a correct NEG result 97% of the time in cases when the disease is not present.

Our hypotheses:

- The patient has the particular cancer
- The patient does not have that particular cancer.

sharpen your pencil

Let's translate what I wrote above into probability notation. Please match up the English statements below with their associated notations and write in the probabilities. If there is no English statement matching a probability, please write one.

We know that only 0.8% of the people in the U.S. have this form of cancer.

$P(POS|cancer)$ = _____

$P(POS|\neg cancer)$ = _____

When the disease is present the test returns a correct POS result 98% of the time;

$P(cancer)$ = _____

$P(\neg cancer)$ = _____

$P(NEG|cancer)$ = _____

it returns a correct NEG result 97% of the time in cases when the disease is not present

$P(NEG|\neg cancer)$ = _____

sharpen your pencil – solution

We know that only 0.8% of the people in the U.S. have this form of cancer.

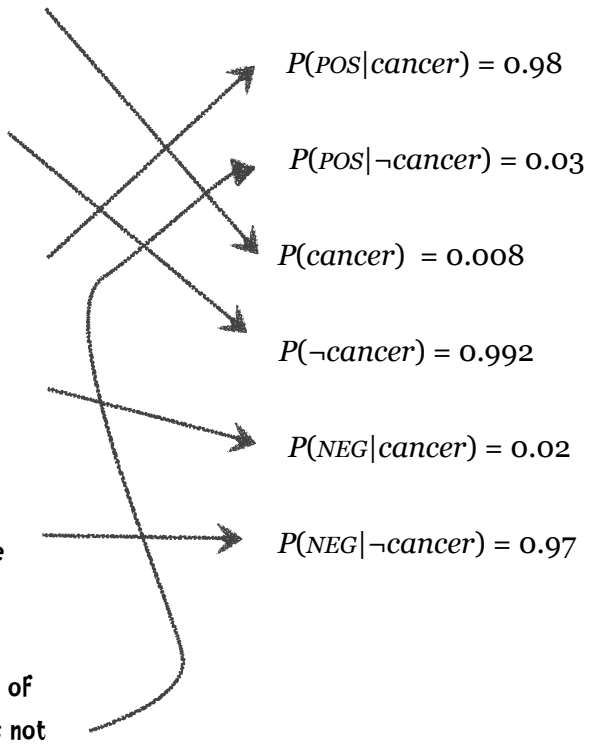
99.2% of people don't have this cancer

When the disease is present the test returns a correct POS result 98% of the time;

When the disease is present the test returns a incorrect NEG result 2% of

it returns a correct NEG result 97% of the time in cases when the disease is not present

it returns an incorrect POS result 3% of the time in cases when the disease is not present



sharpen your pencil – solution

Suppose Ann, comes into the doctor's office

A blood test for cancer is given and the test result is POS.

This is not looking good for Ann. After all, the test is 98% accurate.

Using Bayes Theorem determine whether it is more likely that Ann has cancer or that she does not.

$P(cancer) = 0.008$

$P(\neg cancer) = 0.992$

$P(POS|cancer) = 0.98$

$P(POS|\neg cancer) = 0.03$

$P(NEG|cancer) = 0.02$

$P(NEG|\neg cancer) = 0.97$

sharpen your pencil – solution

Suppose Ann, comes into the doctor's office
A blood test for the cancer is given and the test result is POS.

This is not looking good for Ann. After all, the test is 98% accurate.

Using Bayes Theorem determine whether it is more likely that Ann has cancer or that she does not.

We are finding the maximum a posteriori probability:

$P(POS \mid cancer)P(cancer) = .98(.008) = .0078$

$P(POS \mid \neg\ cancer)\ P(\neg\ cancer) = .03(.992) = .0298$

We select hMAP and classify the patient as not having cancer.

If we want to know the exact probability we can normalize these values by having them sum to 1:

$$P(cancer \mid POS) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

Ann has a 21% chance of having cancer.

You may think "That just doesn't make sense. After all, the test is 98% accurate, but yet you re telling me Ann is most likely not to have cancer. "

You are in good company. 85% of medical doctors get the answer wrong as well.

I just didn't make that 85% number up. See, among others,

Casscells, W., Schoenberger, A., and Grayboys, T. (1978): "Interpretation by physicians of clinical laboratory results." N Engl J Med. 299:999-1001.

Gigerenzer, Gerd and Hoffrage, Ulrich (1995): "How to improve Bayesian reasoning without instruction: Frequency formats." Psychological Review. 102: 684-704.

Eddy, David M. (1982): "Probabilistic reasoning in clinical medicine: Problems and opportunities." In D. Kahneman, P. Slovic, and A. Tversky, eds, Judgement under uncertainty: Heuristics and biases. Cambridge University Press, Cambridge, UK.

Here is why the results seem so counterintuitive. Most people see the statistic that 98% of the people who have this particular cancer will have a positive test result and also conclude that 98% of the people who have a positive test result have this particular cancer. This fails to take into account that this cancer affects only 0.8% of the population. Let's say we give the test to everyone in a city of 1 million people. That means that 8,000 people have cancer and 992,000 do not. First, let's consider giving the test to the 8,000 people with cancer. We know that 98% of the time when we give the test to people with cancer the test correctly returns a positive result. So 7,840 people have a correct positive result and 160 of those people with cancer have an incorrect negative result. Now let's turn to the 992,000 people without cancer. When we give the test to them, 97% of the time we get a correct negative result so (992,000 * 0.97) or 962,240 of them have a correct negative result and 30,000 have an incorrect positive result. I have summarized these results on the following page.

	positive test result	negative test result
people with cancer	7,840	160
people without cancer	30,000	962,240

Now, consider Ann getting a positive test result and the data in the 'positive test result' column. 30,000 of the people with a positive test result had no cancer while only 7,840 of them had cancer. So it seems probable that Ann does not have cancer.

> Still don't get it?
> Don't worry. Many people don't.
> After more practice you will gain a better understanding.

Why do we need Bayes Theorem?

Yet again, Bayes Theorem is

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

Let us return to the shopping cart example presented earlier. In that example, we obtained the information on the right from customers.

Say we know a customer lives in the 88005 zipcode and our two competing hypotheses are that they will buy Sencha tea or they will not. So:

$P(h_1|D) = P(buySenchaTea|88005)$

and

Customer ID	Zipcode	bought organic produce?	bought Sencha green tea?
1	88005	Yes	Yes
2	88001	No	No
3	88001	Yes	Yes
4	88005	No	No
5	88003	Yes	No
6	88005	No	Yes
7	88005	No	No
8	88001	No	No
9	88005	Yes	Yes

$P(h_2|D) = P(\neg\ buySenchaTea|88005)$

In this case you may wonder why we need to compute

$$\frac{P(88005 \mid buySenchaTea)P(buySenchaTea)}{P(88005)}$$

when we can just as easily compute $P(buySenchaTea|88005)$ directly from the data in the table. In this simple case you would be correct but for many real world problems it is very difficult to compute $P(h|D)$ directly.

Consider the previous medical example where we were interested in determining whether a person had cancer or not given that a certain test returned a positive result.

$$P(cancer \mid POS) \approx P(POS \mid cancer)P(cancer)$$

$$P(\neg cancer \mid POS) \approx P(POS \mid \neg cancer)P(\neg cancer)$$

It is relatively easy to compute the items on the right hand side. We can estimate $P(POS|cancer)$ by giving the cancer test to a representative sample of people with cancer and $P(POS|\neg\ cancer)$ by giving the test to a sample of people without cancer. $P(cancer)$ seems like a statistic that would be available on government websites and $P(\neg\ cancer)$ is simply

$1 - P(cancer)$

However, computing $P(cancer|POS)$ directly would be significantly more challenging. This is asking us to determine the probability that when we give the test to a random average person in the entire population and the test result is POS then that person has cancer. To do this we want a representative sample of the population but since only 0.8% of people have cancer a sample size of 1,000 people would only have 8 people with cancer—far too few to feel that our counts are representative of the population as a whole. So we would need an extremely large sample size. So Bayes Theorem provides a strategy for computing $P(h|D)$ when it is hard to do so directly.

Naïve Bayes

Most of the time we have more evidence than just a single piece of data. In the Sencha tea example we had two types of evidence: zip code and whether the person purchased organic food. To compute the probability of an hypothesis given all the evidence, we simply multiply the individual probabilities. In this example

Code:

tea = Person buy Sencha tea

¬ tea = Person does not buy Sencha tea

P(88005|tea) = probability that a person lives in the 88005 zipcode given that person bought Sencha tea.

etc.

Customer ID	Zipcode	bought organic produce?	bought Sencha green tea?
1	88005	Yes	Yes
2	88001	No	No
3	88001	Yes	Yes
4	88005	No	No
5	88003	Yes	No
6	88005	No	Yes
7	88005	No	No
8	88001	No	No
9	88005	Yes	Yes

We would like to know whether a person
who lives in the 88005 zipcode and bought organic produce will likely buy tea:

$P(tea|88005 \ \& \ organic)$ and for that we simply multiply the probabilities:

$P(tea|88005 \ \& \ organic) = P(88005 \mid tea)\ P(organic \mid tea)\ P(tea) = .6(.8)(.5) = .24$

$P(\neg tea|88005 \ \& \ organic) = P(88005 \mid \neg tea)\ P(organic \mid \neg tea)\ P(\neg tea) = .4(.25)(.5) = .05$

So a person who lives in the trendy 88005 zip code area and buys organic food is more likely to buy Sencha Green tea than not. So let's display the Green Tea ad on the shopping cart display!

Here's how Stephen Baker describes the smart shopping cart technology:

> ... here's what shopping with one of these carts might feel like. You grab a cart on the way in and swipe your loyalty card. The welcome screen pops up with a shopping list. It's based on patterns of your last purchases. Milk, eggs, zucchini, whatever. Smart systems might provide you with the quickest route to each item. Or perhaps they'll allow you to edit the list, to tell it, for example, never to promote cauliflower or salted peanuts again. This is simple stuff. But according to Accenture's studies, shoppers forget an average of 11 percent of the items they intend to buy. If stores can effective remind us of what we want, it means fewer midnight runs to the convenience store for us and more sales for them.
>
> Baker. 2008. P49.

The Numerati

I've mentioned this book by Stephen Baker several times. I highly encourage you to read this book. The paperback is only $10 and it is a good late night read.

i100 i500

Let's say we are trying to help iHealth, a company that sells wearable exercise monitors that compete with the Nike Fuel and the Fitbit Flex. iHealth sells two models that increase in functionality: the i100 and the i500:

iHealth100:
heart rate, GPS (to compute miles per hour, etc), wifi to automatically connect to iHealth website to upload data.

iHealth500:
i100 features + pulse oximetry (oxygen in blood) + free 3G connection to iHealth website

They sell these online and they hired us to come up with a recommendation system for their customers. To get data to build our system when someone buys a monitor, we ask them to fill out the questionnaire. Each question in the questionnaire relates to an attribute. First, we ask them what their main reason is for starting an exercise program and have them select among three options: health, appearance or both. We ask them what their current exercise level is: sedentary, moderate, or active. We ask them how motivated they are: moderate or aggressive. And finally we ask them if they are comfortable with using technological devices. Our results are as follows.

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Model #
both	sedentary	moderate	yes	i100
both	sedentary	moderate	no	i100
health	sedentary	moderate	yes	i500
appearance	active	moderate	yes	i500
appearance	moderate	aggressive	yes	i500
appearance	moderate	aggressive	no	i100
health	moderate	aggressive	no	i500
both	active	moderate	yes	i100
both	moderate	aggressive	yes	i500
appearance	active	aggressive	yes	i500
both	active	aggressive	no	i500
health	active	moderate	no	i500
health	sedentary	aggressive	yes	i500
appearance	active	moderate	no	i100
health	sedentary	moderate	no	i100

sharpen your pencil

Using the naïve Bayes method, which model would you recommend to a person whose
main interest is health
current exercise level is moderate
is moderately motivated
and is comfortable with technological devices

Turn the page if you need a hint!

sharpen your pencil clue

Ok. So we want to compute

P(i100 | health, moderateExercise, moderateMotivation, techComfortable)

and

P(i500 | health, moderateExercise, moderateMotivation, techComfortable)

and pick the model with the highest probability.

Let me lay out what we need to do for the first one:

P(i100 | health, moderateExercise, moderateMotivation, techComfortable) =

P(health|i100) P(moderateExercise|i100) P(moderateMotivated|i100)
P(techComfortable|i100)P(i100)

So here is what we need to first compute

P(health|i100) = 1/6 ←

There were 6 occurrences of people buying i100s and only one of those people had a main interest of 'health'

P(moderateExercise|i100) =

P(moderateMotivated|i100) =

P(techComfortable|i100) =

P(i100) = 6 / 15

That was my clue. Now hopefully you can figure out the example

sharpen your pencil solution

First we compute

P(i100 | health, moderateExercise, moderateMotivation, techComfortable)

which equals the product of all these terms:

P(health|i100) P(moderateExercise|i100) P(moderateMotivated|i100)
P(techComfortable|i100)P(i100)

P(health|i100) = 1/6
P(moderateExercise|i100) = 1/6
P(moderateMotivated|i100) = 5/6
P(techComfortable|i100) = 2/6
P(i100) = 6 / 15

so

P(i100| evidence) = .167 * .167 * .833 * .333 * .4 = .**00309**

Now we compute

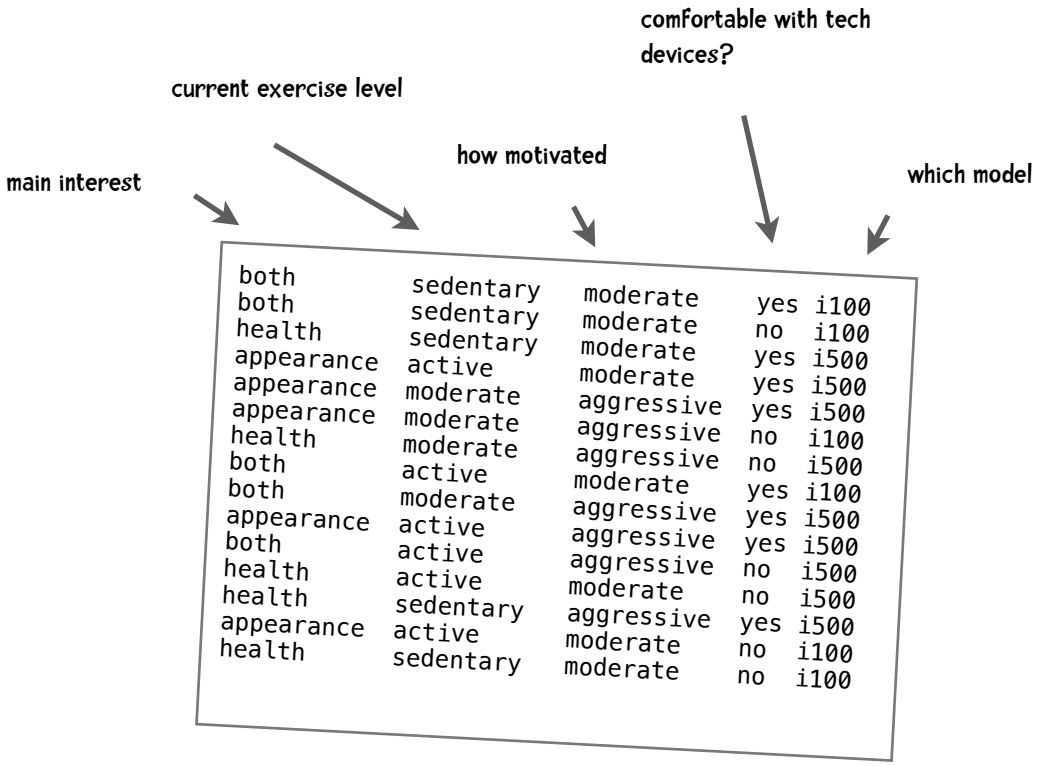
P(i500 | health, moderateExercise, moderateMotivation, techComfortable)

P(health|i500) = 4/9
P(moderateExercise|i500) = 3/9
P(moderateMotivated|i500) = 3/9
P(techComfortable|i500) = 6/9
P(i500) = 9 / 15

P(i500| evidence) = .444 * .333 * .333 * .667 * .6 = .**01975**

Doing it in Python

Great! Now that we understand how a Naïve Bayes Classifier works let us consider how to implement it in Python. The format of the data files will be the same as that in the previous chapter, a text file where each line consists of tab-separated values. For our iHealth example, the data file would look like the following:



Shortly we will be using an example with substantially more data and I would like to keep the ten-fold cross validation methods we used in code from the previous chapter. Recall that that method involved dividing the data into ten buckets (files). We would train on nine of them and test the classifier on the remaining bucket. And we would repeat this ten times; each time withholding a different bucket for testing. The simple iHealth example, with only 15 instances, was designed so we could work through the Naïve Bayes Classifier method by hand. With only 15 instances it seems silly to divide them into 10 buckets. The ad hoc, not very elegant solution we will use, is to have ten buckets but all the 15 instances will be in one bucket and the rest of the buckets will be empty.

The Naïve Bayes Classifier code consists of two components, one for training and one for classifying.

Training

The output of training needs to be:

- a set of prior probabilities—for example, P(i100) = 0.4
- a set of conditional probabilities—for example, P(health|i100) = 0.167

I am going to represent the set of prior probabilities as a Python dictionary (hash table):

```
self.prior = {'i500': 0.6, 'i100': 0.4}
```

The conditional probabilities are a bit more complex. My way of doing this—and there are probably better methods—is to associate a set of conditional probabilities with each class.

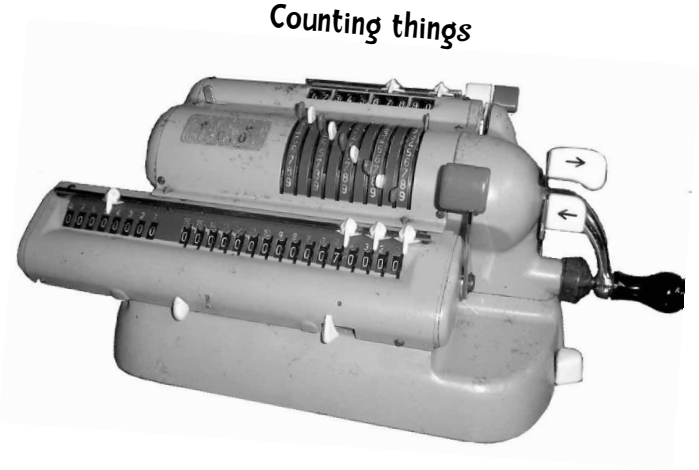
```
{'i500': {1: {'appearance': 0.3333333333333, 'health': 0.4444444444444,
              'both': 0.2222222222222},
          2: {'sedentary': 0.2222222222222, 'moderate': 0.333333333333,
              'active': 0.4444444444444444},
          3: {'moderate': 0.333333333333, 'aggressive': 0.66666666666},
          4: {'no': 0.3333333333333333, 'yes': 0.6666666666666666}},

 'i100': {1: {'appearance': 0.333333333333, 'health': 0.1666666666666,
              'both': 0.5},
          2: {'sedentary': 0.5, 'moderate': 0.16666666666666,
              'active': 0.3333333333333},
          3: {'moderate': 0.83333333334, 'aggressive': 0.166666666666},
          4: {'no': 0.6666666666666, 'yes': 0.3333333333333}}}
```

The 1, 2, 3, 4 represent column numbers. So the first line of the above is "the probability of the value of the first column being 'appearance' given that the device is i500 is 0.333."

The first step in computing these probabilities is simply to count things. Here are the first few lines of the input file:

```
both        sedentary   moderate    yes i100
both        sedentary   moderate    no  i100
health      sedentary   moderate    yes i500
appearance  active      moderate    yes i500
```

Counting things



Yet again I am going to use dictionaries. One, called, `classes`, which will count the occurrences of each class or category. So, after the first line `classes` will look like

```
{'i100': 1}
```

After the second line:

```
{'i100': 2}
```

And after the third:

```
{'i500': 1, 'i100': 2}
```

Prior probability

After I process all the data, the value of `classes` is

```
{'i500': 9, 'i100': 6}
```

To obtain the prior probabilities I simply divide those number by the total number of instances.

Conditional probability

To determine the conditional probabilities I am going to count the occurrences of attribute values in the different columns in a dictionary called `counts`. and I am going to maintain separate counts for each class. So, in processing the string 'both' in the first line, `counts` will be:

```
{'i100': {1: {'both': 1}}
```

and at the end of processing the data, the value of counts will be

```
{'i100': {1: {'appearance':2, 'health': 1, 'both': 3},
          2: {'active': 2, 'moderate': 1, 'sedentary': 3},
          3: {'moderate': 5, 'aggressive': 1},
          4: {'yes': 2, 'no': 4}},
 'i500': {1: {'health': 4, 'appearance': 3, 'both': 2},
          2: {'active': 4, 'moderate': 3, 'sedentary': 2},
          3: {'moderate': 3, 'aggressive': 6},
          4: {'yes': 6, 'no': 3}}}
```

So, in the first column of the i100 instances there were 2 occurrences of 'appearance', 1 of 'health' and 3 of 'both'. To obtain the conditional probabilities we divide those numbers by the total number of instances of that class. For example, there are 6 instances of i100 and 2 of them had a value of 'appearance' for the first column, so

P('appearance'|i100) = 2/6 = .333

With that background here is the Python code for training the classifier (remember, you can download this code at guidetodatamining.com).

```
# _____________________________________________________________________

class BayesClassifier:
    def __init__(self, bucketPrefix, testBucketNumber, dataFormat):

        """ a classifier will be built from files with the bucketPrefix
        excluding the file with textBucketNumber. dataFormat is a
        string that describes how to interpret each line of the data
        files. For example, for the iHealth data the format is:
        "attr     attr  attr  attr  class"
        """

        total = 0
        classes = {}
        counts = {}

        # reading the data in from the file

        self.format = dataFormat.strip().split('\t')
        self.prior = {}
        self.conditional = {}
```

```
# for each of the buckets numbered 1 through 10:
for i in range(1, 11):
    #if it is not the bucket we should ignore, read in the data
    if i != testBucketNumber:
        filename = "%s-%02i" % (bucketPrefix, i)
        f = open(filename)
        lines = f.readlines()
        f.close()
        for line in lines:
            fields = line.strip().split('\t')
            ignore = []
            vector = []
            for i in range(len(fields)):
                if self.format[i] == 'num':
                    vector.append(float(fields[i]))
                elif self.format[i] == 'attr':
                    vector.append(fields[i])
                elif self.format[i] == 'comment':
                    ignore.append(fields[i])
                elif self.format[i] == 'class':
                    category = fields[i]
            # now process this instance
            total += 1
            classes.setdefault(category, 0)
            counts.setdefault(category, {})
            classes[category] += 1
            # now process each attribute of the instance
            col = 0
            for columnValue in vector:
                col += 1
                counts[category].setdefault(col, {})
                counts[category][col].setdefault(columnValue,0)
                counts[category][col][columnValue] += 1
```

```
        #
        # ok done counting. now compute probabilities
        #
        # first prior probabilities p(h)
        #
        for (category, count) in classes.items():
            self.prior[category] = count / total
        #
        # now compute conditional probabilities p(h|D)
        #
        for (category, columns) in counts.items():
              self.conditional.setdefault(category, {})
              for (col, valueCounts) in columns.items():
                  self.conditional[category].setdefault(col, {})
                  for (attrValue, count) in valueCounts.items():
                      self.conditional[category][col][attrValue] = (
                          count / classes[category])
```

That's it for training! No Complex math. Just basic counting!!!

Classifying

Okay, we have trained the classifier. Now we want to classify various instances. For example, which model should we recommend for someone whose primary interest is health is moderately active, moderately motivated, and is comfortable with technology:

```
c.classify(['health', 'moderate', 'moderate', 'yes'])
```

For this we need to compute

$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

When we did this by hand we computing the probability of each hypothesis given the evidence and we simply translate that method to code:

```
def classify(self, itemVector):
    """Return class we think item Vector is in"""
    results = []
    for (category, prior) in self.prior.items():
        prob = prior
        col = 1
        for attrValue in itemVector:
            if not attrValue in self.conditional[category][col]:
                # we did not find any instances of this attribute value
                # occurring with this category so prob = 0
                prob = 0
            else:
               prob = prob * self.conditional[category][col][attrValue]
               col += 1
        results.append((prob, category))
    # return the category with the highest probability
    return(max(results)[1])
```

And when I try the code I get the same results we received when we did this by hand:

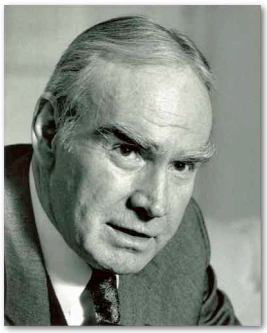
```
>>c = Classifier("iHealth/i", 10, "attr\tattr\tattr\tattr\tclass")
>>print(c.classify(['health', 'moderate', 'moderate', 'yes'])
i500
```

Republicans vs. Democrats

Let us look at a new data set, the Congressional Voting Records Data Set, available from the Machine Learning Repository (http://archive.ics.uci.edu/ml/index.html). It is available in a form that can be used by our programs at http://guidetodatamining.com. The data consists of the voting record of United States Congressional Representatives. The attributes are how that representative voted on 16 different bills.

Attribute Information:

1. Class Name: 2 (democrat, republican)
2. handicapped-infants: 2 (y,n)
3. water-project-cost-sharing: 2 (y,n)
4. adoption-of-the-budget-resolution: 2 (y,n)
5. physician-fee-freeze: 2 (y,n)
6. el-salvador-aid: 2 (y,n)
7. religious-groups-in-schools: 2 (y,n)
8. anti-satellite-test-ban: 2 (y,n)
9. aid-to-nicaraguan-contras: 2 (y,n)
10. mx-missile: 2 (y,n)
11. immigration: 2 (y,n)
12. synfuels-corporation-cutback: 2 (y,n)
13. education-spending: 2 (y,n)
14. superfund-right-to-sue: 2 (y,n)
15. crime: 2 (y,n)
16. duty-free-exports: 2 (y,n)
17. export-administration-act-south-africa: 2 (y,n)



The file consists of tab separated values:

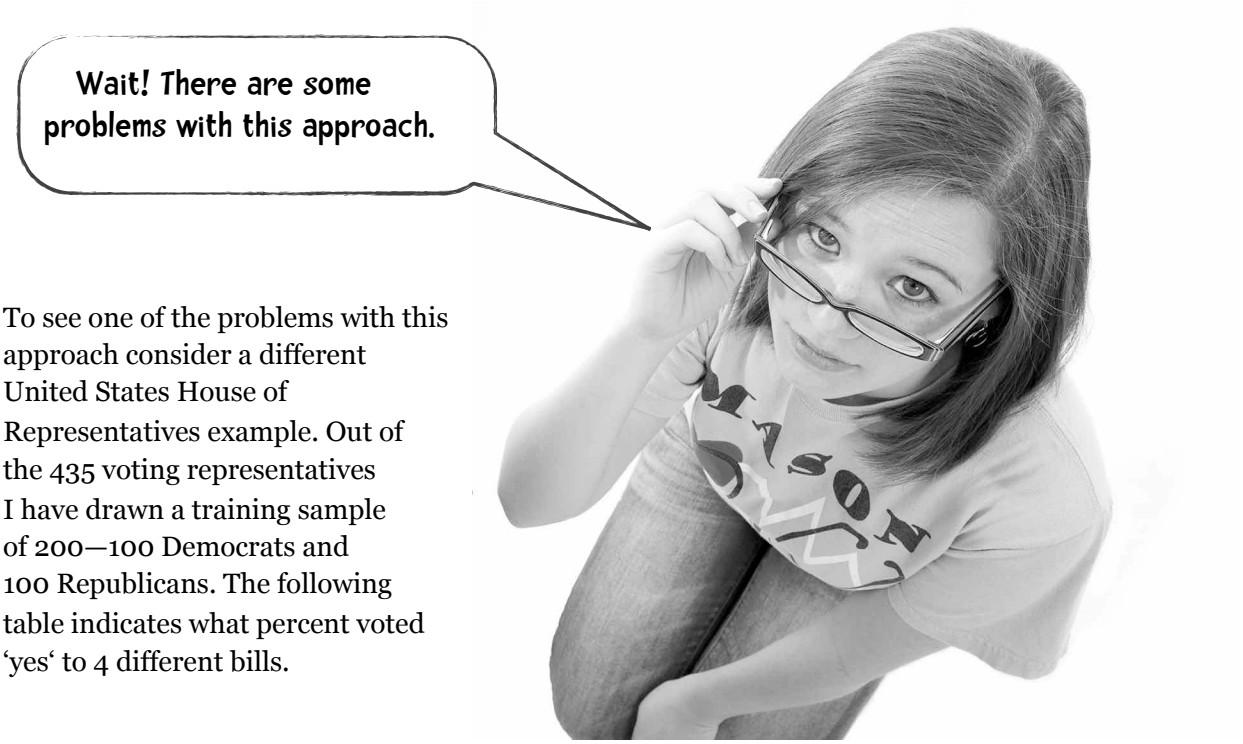
democrat	y	n	y	n	n	n	y	y	y	n	n	n	n	n	y	y
democrat	y	y	y	n	n	n	y	y	y	y	n	n	n	n	y	y
democrat	y	y	y	n	n	n	y	y	n	n	n	n	n	y	n	y
republican	y	y	y	n	n	y	y	y	y	y	n	n	n	n	n	y

Our Naïve Bayes Classifier works fine with this example (the format string says that the first column is to be interpreted as the class of the instance and the rest of the columns are to be interpreted as attributes):

```
format = """class\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tattr
\tattr\tattr\tattr\tattr\tattr\tattr\tattr"""

tenfold("house-votes/hv", format)
```

```
            Classified as:
               democrat   republican
               +-------+-------+
   democrat    |   111 |    13 |
               |-------+-------|
 republican    |     9 |    99 |
               +-------+-------+

90.517 percent correct
total of 232 instances
```

That's great!



To see one of the problems with this approach consider a different United States House of Representatives example. Out of the 435 voting representatives I have drawn a training sample of 200—100 Democrats and 100 Republicans. The following table indicates what percent voted 'yes' to 4 different bills.

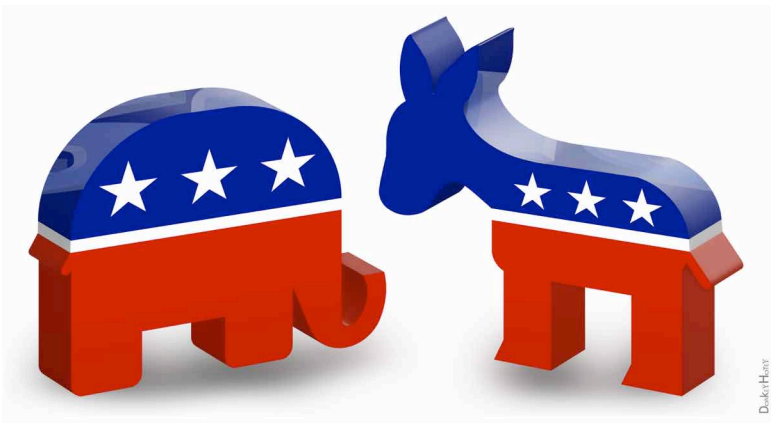
	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Republican	0.99	0.01	0.99	0.5
Democrat	0.01	0.99	0.01	1.0

% voting 'yes'

This table shows that 99% of Republicans in the sample voted for the CISPA (Cyber Intelligence Sharing and Protection Act), only 1% voted for the Reader Privacy Act, 99% voted for Internet Sales Tax and 50% voted for the Internet Snooping Bill. (I made up these numbers and they do not reflect reality.) We pick a U.S. Representative who wasn't in our sample, Representative X, who we would like to classify as either a Democrat or Republican. I added how that representative voted to our table:

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Republican	0.99	0.01	0.99	0.5
Democrat	0.01	0.99	0.01	1.0
Rep. X	N	Y	N	N

Do you think the person is a Democrat or Republican?



I would guess Democrat. Let us work through the example step-by-step using Naïve Bayes. The prior probabilities of *P*(*Democrat*) and *P*(*Republican*) are both 0.5 since there are 100 Republicans and 100 Democrats in the sample. We know that Representative X voted 'no' to CISPA and we also know

P(Republican|C=no) = 0.01 and P(Democrat|C=no) = 0.99

where C = CISPA. And with that bit of evidence our current P(h|D) probabilities are

h=	p(h)	P(C=no|h)				P(h|D)
Republican	0.5	0.01				0.005
Democrat	0.5	0.99				0.495

Factoring in Representative X's 'yes' vote to the Reader Privacy Act and X's 'no' to the sales tax bill we get:

h=	p(h)	P(C=no|h)	P(R=yes|h)	P(T=no|h)		P(h|D)
Republican	0.5	0.01	0.01	0.01		0.0000005
Democrat	0.5	0.99	0.99	0.99		0.485

If we normalize these probabilities:

$$P(Democrat \mid D) = \frac{0.485}{0.485 + 0.0000005} = \frac{0.485}{0.4850005} = 0.99999$$

So far we are 99.99% sure Representative X is a Democrat.

Finally, we factor in Representative X's 'no' vote on the Internet Snooping Bill.

h=	p(h)	P(C=no|h)	P(R=yes|h)	P(T=no|h)	P(S=no|h)	P(h|D)
Republican	0.5	0.01	0.01	0.01	0.50	2.5E-07
Democrat	0.5	0.99	0.99	0.99	0.00	0

Whoops. We went from 99.99% likelihood that X was a Democrat to 0%. This is so because we had 0 occurrences of a Democrat voting 'no' for the snooping bill.

Estimating Probabilities

The probabilities in Naïve Bayes are really **estimates** of the true probabilities. True probabilities are those obtained from the entire population. For example, if we could give a cancer test to everyone in the entire population, we could, for example, get the true probability of the test returning a negative result given that the person does not have cancer. However, giving the test to everyone is near impossible. We can estimate that probability by selecting a random representative sample of the population, say 1,000 people, giving the test to them, and computing the probabilities. Most of the time this gives us a very good estimate of the true probabilities, but when the true probabilities are very small, these estimates are likely to be poor. Here is an example. Suppose the true probability of a Democrat voting no to the Internet Snooping Bill is 0.03—$P(S=no|Democrat) = 0.03$.

Brain Calisthenics

Suppose we try to estimate these probabilities by selected a sample of 10 Democrats and 10 Republicans. What is the most probable number of Democrats in the sample that voted no to the snooping bill?

☐0 ☐1 ☐2 ☐3

Brain Calisthenics—answer

Suppose we try to estimate these probabilities by selected a sample of 10 Democrats and 10 Republicans. What is the most probable number of Democrats in the sample that voted no to the snooping bill?

0

So based on the sample P(S=no|Democrat) = 0.

As we just saw in the previous example, when a probability is 0 it dominates the Naïve Bayes calculation—it doesn't matter what the other values are. Another problem is that probabilities based on a sample produce a biased underestimate of the true probability.

Fixing this.

If we a trying to calculate something like P(S=no|Democrat) our calculation has been

$$P(\text{S=no}|\text{Democrat}) = \frac{\text{the number that both are Democrats and voted no on the snooping bill.}}{\text{total number of Democrats}}$$

For expository ease let me simplify this by using shorter variable names:

$$P(x \mid y) = \frac{n_c}{n}$$

Here n is the total number of instances of class y in the training set; n_c is the total number of instances of class y that have the value x.

The problem we have is when n_c equals zero. We can eliminate this problem by changing the formula to:

$$P(x \mid y) = \frac{n_c + mp}{n + m}$$

This formula is from p179 of the book "Machine Learning" by Tom Mitchell.

m is a constant called the equivalent sample size. The method for determining the value of m varies. For now I will use the number of different values that attribute takes. For example, there are 2 values for how a person voted on the snooping bill, *yes*, or *no*. So I will use an m of 2. p is the prior estimate of the probability. Often we assume uniform probability. For example, what is the probability of someone voting no to the snooping bill knowing nothing about that person? 1/2. So p in this case is 1/2.

Let's go through the previous example to see how this works. First, here are tables showing the vote:

Republican Vote

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Yes	99	1	99	50
No	1	99	1	50

Democratic Vote

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Yes	1	99	1	0
No	99	1	99	100

The person we are trying to classify voted no to CISPA. First we compute the probability that he's a Republican given that vote. Our new formula is

$$P(x \mid y) = \frac{n_c + mp}{n + m}$$

n is the number or Republicans which is 100 and n_c is the number of Republicans who voted no to CISPA which is 1. m is the number of values for the attribute "how they voted on CISPA", which is 2 (*yes* or *no*). So plugging those number into our formula

$$P(cispa = no \mid republican) = \frac{1 + 2(.5)}{100 + 2} = \frac{2}{102} = 0.01961$$

We follow the same procedure for a person voting no to CISPA given they are a Democrat.

$$P(cispa = no \mid democrat) = \frac{99 + 2(.5)}{100 + 2} = \frac{100}{102} = 0.9804$$

With that bit of evidence our current P(h|D) probabilities are

h=	p(h)	P(C=no|h)				P(h|D)
Republican	0.5	0.01961				0.0098
Democrat	0.5	0.9804				0.4902

Factoring in Representative X's 'yes' vote to the Reader Privacy Act and X's 'no' to the sales

sharpen your pencil

Finish this problem and classify the individual as either a Republican or Democrat.

Recall, he voted no to Cispa, yes to the Reader Privacy act, and no both to the sales tax and snooping bills.

sharpen your pencil -answer

Finish this problem and classify the individual as either a Republican or Democrat.

Recall, he voted no to CISPA, yes to the Reader Privacy act, and no both to the Internet sales tax and snooping bills.

The calculations for the next 2 columns mirror those we did for the CISPA vote. The probability that this person voted no to the snooping bill given that he is a Republican is

$$P(s = no \mid republican) = \frac{50 + 2(.5)}{100 + 2} = \frac{51}{102} = 0.5$$

and that he voted no given that he is a Democrat:

$$P(s = no \mid democrat) = \frac{0 + 2(.5)}{100 + 2} = \frac{1}{102} = 0.0098$$

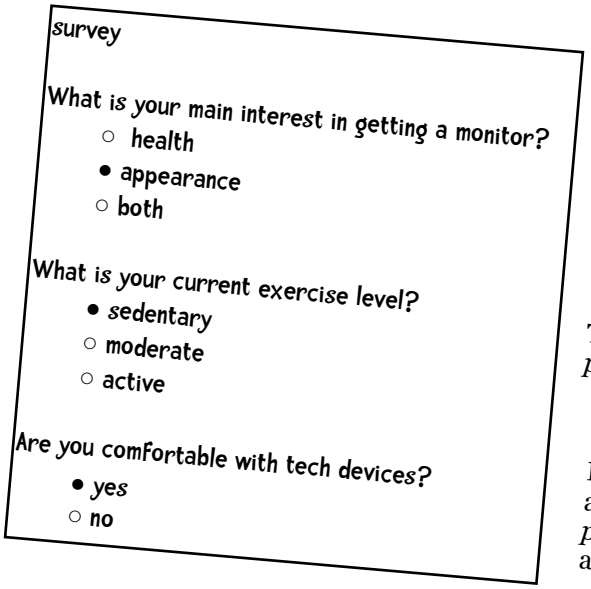
Multiplying those probabilities together gives us

h=	p(h)	P(C=no|h)	P(R=yes|h)	P(I=no|h)	P(S=no|h)	P(h|D)
Republican	0.5	0.01961	0.01961	0.01961	0.5	0.000002
Democrat	0.5	0.9804	0.9804	0.9804	0.0098	0.004617

So unlike the previous approach we would classify this individual as a Democrat. This matches our intuitions.

A clarification

For this example, the value of m was 2 for all calculations. However, it is not the case that m remains necessarily constant across attributes. Consider the health monitor example discussed earlier in the chapter. The attributes for that example included:



For this attribute, $m = 3$ since the attribute can take one of 3 values (health, appearance, both). If we assume uniform probabilities, then $p = 1/3$ since the probability of the attribute being any one of the values is

This attribute also has $m = 3$ and $p = 1/3$

For this attribute, $m = 2$ since the attribute can take one of 2 values and $p = 1/2$ since the probability of the attribute being any one of those is 1/2

Let us say the number of the people surveyed who own the i500 monitor is 100 (this is n). The number of people who own a i500 and are sedentary is 0 (n_c). So, the probability of someone being sedentary given they own an i500 is

$$P(sedentary \mid i500) = \frac{n_c + mp}{n + m} = \frac{0 + 3(.333)}{100 + 3} = \frac{1}{103} = 0.0097$$

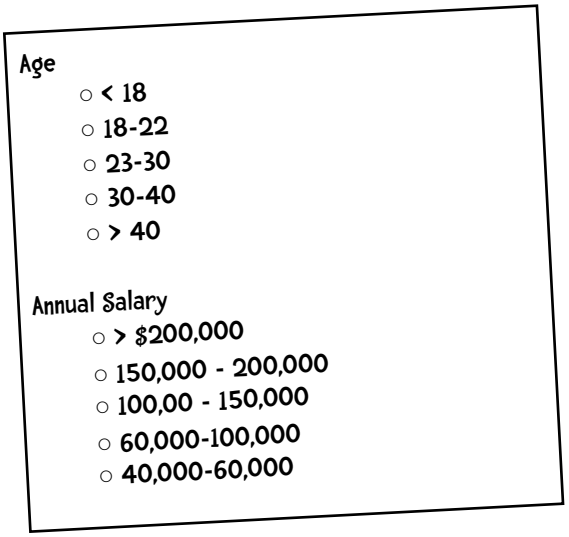
Numbers

You probably noticed that I changed from **numerical** data which I used in all the nearest neighbor approaches I discussed to using **categorical** data for the naïve Bayes formula. By "categorical data" we mean that the data is put into discrete categories. For example, we divide people up in how they voted for a particular bill and the people who voted 'yes' go in one category and the people who voted 'no' go in another. Or we might categorize musicians by the instrument they play. So all saxophonists go in one bucket, all drummers in another, all pianists in another and so on. And these categories do not form a scale. So, for example, saxophonists are not 'closer' to pianists than they are to drummers. Numerical data is on a scale. An annual salary of $105,000 is closer to a salary of $110,000 than it is to one of $40,000.

For Bayesian approaches we count things—how many occurrences are there of people who are sedentary—and it may not be initially obvious how to count things that are on a scale—for example, something like grade point average. There are two approaches.

Method 1: Making categories

One solution is to make categories by discretizing the continuous attribute. You often see this on websites and on survey forms. For example:



Once we have this information divided nicely into discrete values, we can use Naïve Bayes exactly as we did before.

Method 2: Gaussian distributions!

The terms "Gaussian Distribution" and "Probability Density Function" sound cool, but they are more than good phrases to know so you can impress your friends at dinner parties. So what do they mean and how they can be used with the Naïve Bayes method? Consider the example we have been using with an added attribute of income:

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Income (in $1,000)	Model #
both	sedentary	moderate	yes	60	i100
both	sedentary	moderate	no	75	i100
health	sedentary	moderate	yes	90	i500
appearance	active	moderate	yes	125	i500
appearance	moderate	aggressive	yes	100	i500
appearance	moderate	aggressive	no	90	i100
health	moderate	aggressive	no	150	i500
both	active	moderate	yes	85	i100
both	moderate	aggressive	yes	100	i500
appearance	active	aggressive	yes	120	i500
both	active	aggressive	no	95	i500
health	active	moderate	no	90	i500
health	sedentary	aggressive	yes	85	i500
appearance	active	moderate	no	70	i100
health	sedentary	moderate	no	45	i100

Let's think of the typical purchaser of an i500, our awesome, premiere device. If I were to ask you to describe this person you might give me the average income:

$$mean = \frac{90+125+100+150+100+120+95+90+85}{9} = \frac{955}{9} = 106.111$$

And perhaps after reading chapter 4 you might give the standard deviation:

$$sd = \sqrt{\frac{\sum_i (x_i - \overline{x})^2}{card(x)}}$$

Recall that the standard deviation describes the range of scattering. If all the values are bunched up around the mean, the standard deviation is small; if the values are scattered the standard deviation is large

sharpen your pencil

What is the income standard deviation of the people who bought the i500? (those values are shown in the column below)

Income (in $1,000)
90
125
100
150
100
120
95
90
85

sharpen your pencil - solution

What is the standard deviation of the income of the people who bought the i500? (those values are shown in the column above)

Income (in $1,000)	(x-106.111)	(x-106.111)^2
90	-16.111	259.564
125	18.889	356.794
100	-6.111	37.344
150	43.889	1926.244
100	-6.111	37.344
120	13.889	192.904
95	-11.111	123.454
90	-16.111	259.564
85	-21.111	445.674

$\sum$ = 3638.889

$$sd = \sqrt{\frac{3638.889}{9}}$$

$$= \sqrt{404.321} = 20.108$$

Population standard deviation and sample standard deviation.

The formula for standard deviation that we just used is called the population standard deviation. It is called that because we use this formula when we have data on the entire population we are interested in. For example, we might give a test to 500 students and then compute the mean and standard deviation. In this case, we would use the population standard deviation, which is what we have been using. Often, though, we do not have data on the entire population. For example, suppose I am interested in the effects of drought on the deer mice in Northern New Mexico and as part of that study I want the average (mean) and standard deviation of their weights. In this case I am not going to weigh every mouse in Northern New Mexico. Rather I will collect and weigh some representative sample of mice.

For this sample, I can use the standard deviation formula I used above, but there is another formula that has been shown to be a better estimate of the entire population standard deviation. This formula is called the **sample standard deviatio**n and it is just a slight modification of the previous formula:

$$sd = \sqrt{\frac{\sum_i (x_i - \overline{x})^2}{card(x) - 1}}$$

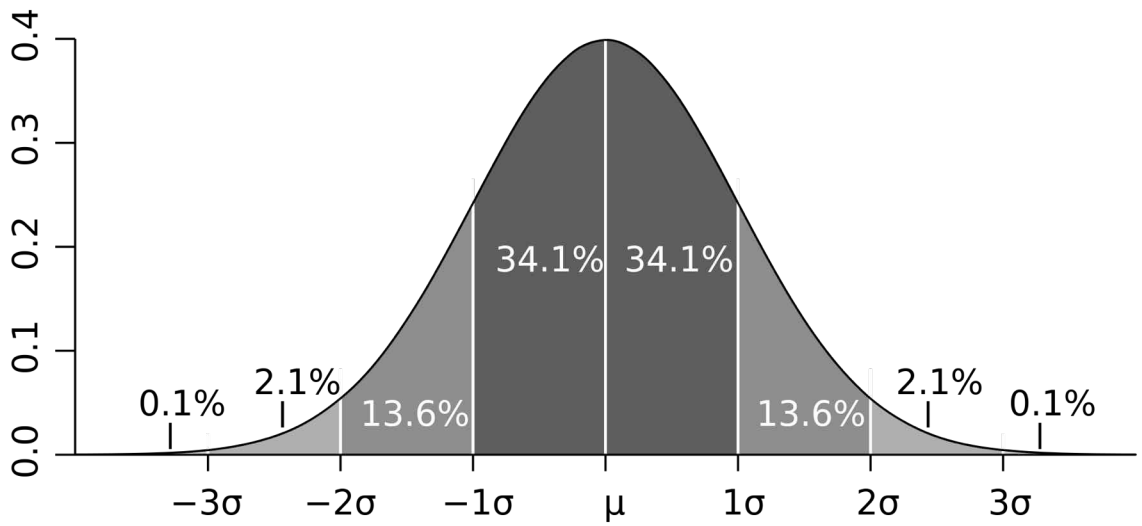
The sample standard deviation of the income example is

$$sd = \sqrt{\frac{3638.889}{9-1}} = \sqrt{\frac{3638.889}{8}}$$

$$= \sqrt{454.861} = 21.327$$

For the rest of this chapter we will be using sample standard deviation.

You probably have heard terms such as normal distribution, bell curve, and Gaussian distribution. Gaussian distribution is just a high falutin term for normal distribution. The function that describes this distribution is called the Gaussian function or bell curve. Most of the time the Numerati (aka data miners) assume attributes follow a Gaussian distribution. What is means is that about 68% of the instances in a Gaussian distribution fall within 1 standard deviation of the mean and 95% of the instances fall within 2 standard deviations of the mean:



In our case, the mean was 106.111 and the sample standard deviation was 21.327. So 95% of the people who purchase an i500 earn between $42,660 and $149,770. If I asked you if you thought P(100k| i500) —the likelihood that an i500 purchaser earns $100,000—was, you might think that's pretty likely. If I asked you what you thought the likelihood of P(20k| i500)—the likelihood that an i500 purchaser earns $20,000—was , you might think it was pretty unlikely.

To formalize this, we are going to use the mean and standard deviation to compute this probability as follows:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

Maybe putting the formula in a bigger font makes it look simpler!

Everytime I type a complex looking formula into this book, I feel the need to say something like "don't panic." It could be that none of you readers panic and I am just the one panicking.

However, let me say this. Data mining has professional terminology and formulas. Before you dive into data mining you might think "those things look difficult." But after you study, even for a short time, these formulas become nothing special. It is just a matter of working through the formula out step-by-step.

Let's jump right into dissecting this formula so we can see how simple it really is. Let us say we are interested in computing P(100k|i500), the probability that a person earns $100,000 (or 100k) given they purchased an i500. A few pages ago we computed the average income (mean) of people who bought the i500. We also computed the sample standard deviation. These values are shown on the following page. In Numerati speak, we represent the mean with the Greek letter μ (mu) and the standard deviation as σ (sigma).

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

$\mu_{ij} = 106.111$
$\sigma_{ij} = 21.327$
$x_i = 100$

Let's plug these values into the formula:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}(21.327)} e^{\frac{-(100-106.111)^2}{2(21.327)^2}}$$

and do some math:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{6.283}(21.327)} e^{\frac{-(37.344)}{909.68}}$$

and more math:

$$P(x_i \mid y_j) = \frac{1}{53.458} e^{-0.0411}$$

The *e* is a mathematical constant that is the base of the natural logarithm. It's value is approximately 2.718.

$$P(x_i \mid y_j) = \frac{1}{53.458}(2.718)^{-0.0411} = (0.0187)(0.960) = 0.0180$$

So the probability that the income of a person who bought the i500 is $100,000 is 0.0180.

sharpen your pencil

In the table below I have the horsepower ratings for cars that get 35 miles per gallon. I would like to know the probability of a Datsun 280z having 132 horsepower given it gets 35 miles per gallon.

car	HP
Datsun 210	65
Ford Fiesta	66
VW Jetta	74
Nissan Stanza	88
Ford Escort	65
Triumph tr7 coupe	88
Plymouth Horizon	70
Suburu DL	67

μ_{ij} = _____

σ_{ij} = _____

x_i = _____

sharpen your pencil -solution - part 1

In the table below I have the horsepower ratings for cars that get 35 miles per gallon. I would like to know the probability of a Datsun 280z having 132 horsepower given it gets 35 miles per gallon.

car	HP
Datsun 210	65
Ford Fiesta	66
VW Jetta	74
Nissan Stanza	88
Ford Escort	65
Triumph tr7 coupe	88
Plymouth Horizon	70
Suburu DL	67

μ_{ij} = 72.875

σ_{ij} = 9.804

x_i =132

$$\sigma = \sqrt{\frac{(65-\mu)^2 +(66-\mu)^2 + (74-\mu)^2 + (88-\mu)^2 + (65-\mu)^2 + (88-\mu)^2 + (70-\mu)^2 + (67-\mu)^2}{7}}$$

$$\sigma = \sqrt{\frac{672.875}{7}} = \sqrt{96.126} = 9.804$$

sharpen your pencil -solution - part 2

In the table below I have the horsepower ratings for cars that get 35 miles per gallon. I would like to know the probability of a Datsun 280z having 132 horsepower given it gets 35 miles per gallon.

μ_{ij} = 72.875

σ_{ij} = 9.804

x_i =132

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

$$P(132hp \mid 35mpg) = \frac{1}{\sqrt{2\pi}(9.804)} e^{\frac{-(132-72.875)^2}{2(9.804)^2}}$$

$$= \frac{1}{\sqrt{6.283}(9.804)} e^{\frac{-3495.766}{192.237}} = \frac{1}{24.575} e^{-18.185}$$

$$= 0.0407(0.00000001266)$$

$$= 0.0000000005152$$

Ok. it is extremely unlikely that a Datsun 280z, given that it gets 35 miles to the gallon has 132 horsepower. (but it does!)

A few implementation notes.

In the training phase for Naive Bayes, we will compute the mean and sample standard deviation of every numeric attribute. Shortly, we will see how to do this in detail.

In the classification phase, the above formula can be implemented with just a few lines of Python:

```
import math

def pdf(mean, ssd, x):
   """Probability Density Function  computing P(x|y)
   input is the mean, sample standard deviation for all the items in y,
   and x."""
   ePart = math.pow(math.e, -(x-mean)**2/(2*ssd**2))
   return (1.0 / (math.sqrt(2*math.pi)*ssd)) * ePart
```

Let's test this with the examples we did above:

```
>>>pdf(106.111, 21.327, 100)
0.017953602706962717

>>>pdf(72.875, 9.804, 132)
5.152283971078022e-10
```

Whew! Time for a break!

Python Implementation

Training Phase

The Naïve Bayes method relies on prior and conditional probabilities. Let's go back to our Democrat/Republican example. Prior probabilities are the probabilities that hold before we have observed any evidence. For example, if I know there are 233 Republicans and 200 Democrats, then the prior probability of some arbitrary member of the U.S. House of Representatives being a Republican is

$$P(republican) = \frac{233}{433} = 0.54$$

This is denoted $P(h)$. Conditional Probability $P(h|D)$ is the probability that h holds given that we know D, for example, $P(democrat|bill1Vote=yes)$. In Naïve Bayes, we flip that probability and compute P(D|h)—for example, $P(bill1Vote=yes|democrat)$.

In our existing Python program we store these conditional probabilities in a dictionary of the following form:

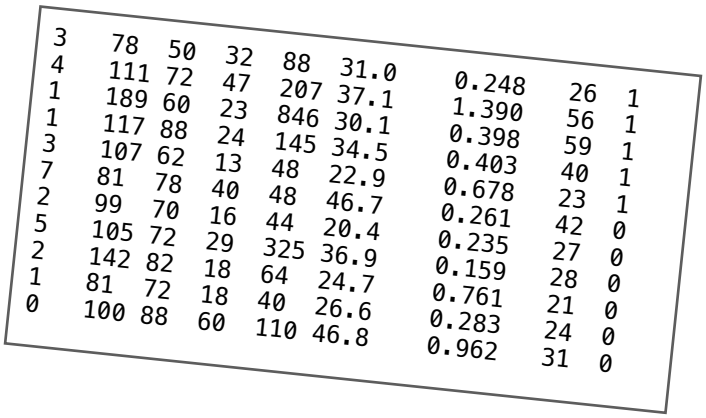
```
{'democrat': {'bill 1': {'yes': 0.333, 'no': 0.667},
              'bill 2': {'yes': 0.778, 'moderate': 0.222}}

 'republican': {'bill 1': {'yes': 0.811, 'no': 0.189},
                'bill 2': {'yes': 0.250, 'no': 0.750}}}
```

So the probability that someone voted yes to bill 1 given that they are a Democrat ($P(bill\ 1=yes|democrat)$) is 0.667.

We will keep this data structure for attributes whose values are discrete values (for example, 'yes', 'no', 'sex=male', 'sex=female'). However, when attributes are numeric we will be using the probability density function and we need to store the mean and sample standard deviation for that attribute. For these numeric attributes I will use the following structures:

```
mean = {'democrat': {'age': 57,  'years served': 12}
        'republican': {'age': 53, 'years served': 7}}

ssd = {'democrat': {'age': 7,  'years served': 3}
       'republican': {'age': 5, 'years served': 5}}
```

As before, each instance is represented by a line in a data file. The attributes of each instances are separated by tabs. For example, the first few lines of a data file for the Pima Indians Diabetes Data set might be:



The columns represent, in order, the number of times pregnant, plasma glucose concentration, blood pressure, triceps skin fold thickness, serum insulin level, body mass index, diabetes pedigree function, age, and a '1' in the last column represents that they developed diabetes and a '0' they did not.

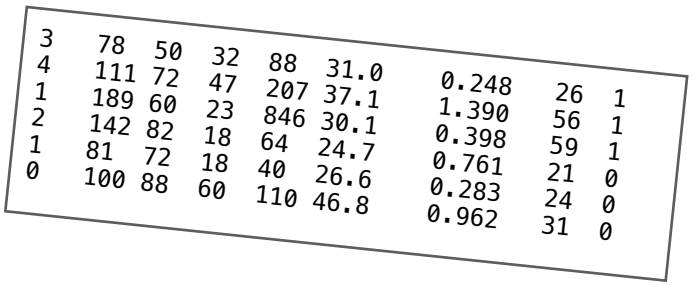
Also as before, we are going to represent how the program should interpret each column by use of a format string, which uses the terms

- `attr` identifies columns that should be interpreted as non-numeric attributes, and which will use the Bayes methods shown earlier in this chapter.
- `num` identifies columns that should be interpreted as numeric attributes, and which will use the Probability Density Function (so we will need to compute the mean and standard deviation during training).
- `class` identifies the column representing the class of the instance (what we are trying to learn)

In the Pima Indian Diabetes data set the format string will be

```
"num    num     num     num    num    num    num    num    class"
```

To compute the mean and sample standard deviation we will need some temporary data structures during the training phase. Again, let us look at a small sample of the Pima data set.



```
3   78  50  32  88  31.0    0.248   26  1
4   111 72  47  207 37.1    1.390   56  1
1   189 60  23  846 30.1    0.398   59  1
2   142 82  18  64  24.7    0.761   21  0
1   81  72  18  40  26.6    0.283   24  0
0   100 88  60  110 46.8    0.962   31  0
```

The last column represents the class of each instance. So the first three individuals developed diabetes and that last three did not. All the other columns represent numeric attributes. of which we need to compute the mean and standard deviation for each of the two classes. To compute the mean for each class and attribute I will need to keep track of the running total. In our existing code we already keep track of the total number of instances. I will implement this using a dictionary:

```
totals   {'1': {1: 8, 2: 378, 3: 182, 4: 102, 5: 1141,
              6: 98.2, 7: 2.036, 8: 141},

         {'0': {1: 3, 2: 323, 3: 242, 4: 96, 5: 214,
              6: 98.1, 7: 2.006, 8: 76}
```

So for class 1, the column 1 total is 8 (3 + 4 + 1), the column 2 total is 378, etc.

For class 0, the column 1 total is 3 (2 + 1 + 0), the column 2 total is 323 and so on.

For standard deviation, we will also need to keep the original data, and for that we will use a dictionary in the following format:

`numericValues`

```
{'1': 1: [3, 4, 1], 2: [78, 111, 189], ...},
{'0': {1: [2, 1, 0], 2: [142, 81, 100]}
```

I have added the code to create these temporary data structures to the `__init__()` method of our Classifier class as shown below:

```
import math

class Classifier:
    def __init__(self, bucketPrefix, testBucketNumber, dataFormat):

        """ a classifier will be built from files with the bucketPrefix
        excluding the file with textBucketNumber. dataFormat is a string that
        describes how to interpret each line of the data files. For example,
        for the iHealth data the format is:
        "attr attr   attr   attr   class"
        """
        total = 0
        classes = {}
        # counts used for attributes that are not numeric
        counts = {}
        # totals used for attributes that are numereric
        # we will use these to compute the mean and sample standard deviation
        # for  each attribute - class pair.
        totals = {}
        numericValues = {}

        # reading the data in from the file
        self.format = dataFormat.strip().split('\t')
        #
        self.prior = {}
        self.conditional = {}

        # for each of the buckets numbered 1 through 10:
        for i in range(1, 11):
            # if it is not the bucket we should ignore, read in the data
            if i != testBucketNumber:
                filename = "%s-%02i" % (bucketPrefix, i)
                f = open(filename)
```

```
lines = f.readlines()
f.close()
for line in lines:
    fields = line.strip().split('\t')
    ignore = []
    vector = []
    nums = []
    for i in range(len(fields)):
        if self.format[i] == 'num':
            nums.append(float(fields[i]))
        elif self.format[i] == 'attr':
            vector.append(fields[i])
        elif self.format[i] == 'comment':
            ignore.append(fields[i])
        elif self.format[i] == 'class':
            category = fields[i]
    # now process this instance
    total += 1
    classes.setdefault(category, 0)
    counts.setdefault(category, {})
    totals.setdefault(category, {})
    numericValues.setdefault(category, {})
    classes[category] += 1
    # now process each non-numeric attribute of the instance
    col = 0
    for columnValue in vector:
        col += 1
        counts[category].setdefault(col, {})
        counts[category][col].setdefault(columnValue, 0)
        counts[category][col][columnValue] += 1
    # process numeric attributes
    col = 0
    for columnValue in nums:
        col += 1
        totals[category].setdefault(col, 0)
        #totals[category][col].setdefault(columnValue, 0)
        totals[category][col] += columnValue
        numericValues[category].setdefault(col, [])
        numericValues[category][col].append(columnValue)
```

```
        #
        # ok done counting. now compute probabilities
        # first prior probabilities p(h)
        #
        for (category, count) in classes.items():
            self.prior[category] = count / total
        #
        # now compute conditional probabilities p(h|D)
        #
        for (category, columns) in counts.items():
              self.conditional.setdefault(category, {})
              for (col, valueCounts) in columns.items():
                  self.conditional[category].setdefault(col, {})
                  for (attrValue, count) in valueCounts.items():
                      self.conditional[category][col][attrValue] = (
                          count / classes[category])
        self.tmp =  counts
        #
        # now compute mean and sample standard deviation
        #
```

code it

Can you add the code to compute the means and standard deviations? Download the file naiveBayesDensityFunctionTraining.py from guidetodatamining.com.

Your program should produce the data structures ssd and means:

```
c = Classifier("pimaSmall/pimaSmall",  1,
               "num num    num    num    num    num    num    num    class")
>> c.ssd
{'0': {1: 2.54694671925252, 2: 23.454755259159146,  ...},
 '1': {1: 4.21137914295475, 2: 29.52281872377408,}}
>>> c.means
{'0': {1: 2.8867924528301887, 2: 111.90566037735849,  ...},
 '1': {1: 5.25, 2: 146.05555555555554, ...}}
```

code it solution

Here is my solution:

```
        #
        # now compute mean and sample standard deviation
        #
        self.means = {}
        self.ssd = {}
        self.totals = totals
        for (category, columns) in totals.items():
            self.means.setdefault(category, {})
            for (col, cTotal) in columns.items():
                self.means[category][col] = cTotal / classes[category]
        # standard deviation

        for (category, columns) in numericValues.items():

            self.ssd.setdefault(category, {})
            for (col, values) in columns.items():
                SumOfSquareDifferences = 0
                theMean = self.means[category][col]
                for value in values:
                    SumOfSquareDifferences += (value - theMean)**2
                columns[col] = 0
                self.ssd[category][col] = math.sqrt(SumOfSquareDifferences
                                            / (classes[category]  - 1))
```

The file containing this solution is naiveBayesDensityFunctionTrainingSolution.py at our website.

code it 2

Can you revise the classify method so it uses the probability density function for numeric attributes? The file to modify is naiveBayesDensityFunctionTemplate.py. Here is the original classify method:

```
def classify(self, itemVector, numVector):
        """Return class we think item Vector is in"""
        results = []
        sqrt2pi = math.sqrt(2 * math.pi)
        for (category, prior) in self.prior.items():
            prob = prior
            col = 1
            for attrValue in itemVector:
                if not attrValue in self.conditional[category][col]:
                   # we did not find any instances of this attribute value
                   # occurring with this category so prob = 0
                   prob = 0
                else:
                   prob = prob * self.conditional[category][col][attrValue]
                col += 1
        # return the category with the highest probability
        #print(results)
        return(max(results)[1])
```

code it 2 - solution

Can you revise the classify method so it uses the probability density function for numeric attributes? The file to modify is naiveBayesDensityFunctionTemplate.py.

Solution:

```
  def classify(self, itemVector, numVector):
        """Return class we think item Vector is in"""
        results = []
        sqrt2pi = math.sqrt(2 * math.pi)
        for (category, prior) in self.prior.items():
            prob = prior
            col = 1
            for attrValue in itemVector:
                if not attrValue in self.conditional[category][col]:
                   # we did not find any instances of this attribute
value
                   # occurring with this category so prob = 0
                   prob = 0
                else:
                   prob = prob * self.conditional[category][col]
[attrValue]
                col += 1
            col = 1
            for x in  numVector:
                mean = self.means[category][col]
                ssd = self.ssd[category][col]
                ePart = math.pow(math.e, -(x - mean)**2/(2*ssd**2))
                prob = prob * ((1.0 / (sqrt2pi*ssd)) * ePart)
                col += 1
            results.append((prob, category))
        # return the category with the highest probability
        #print(results)
        return(max(results)[1])
```

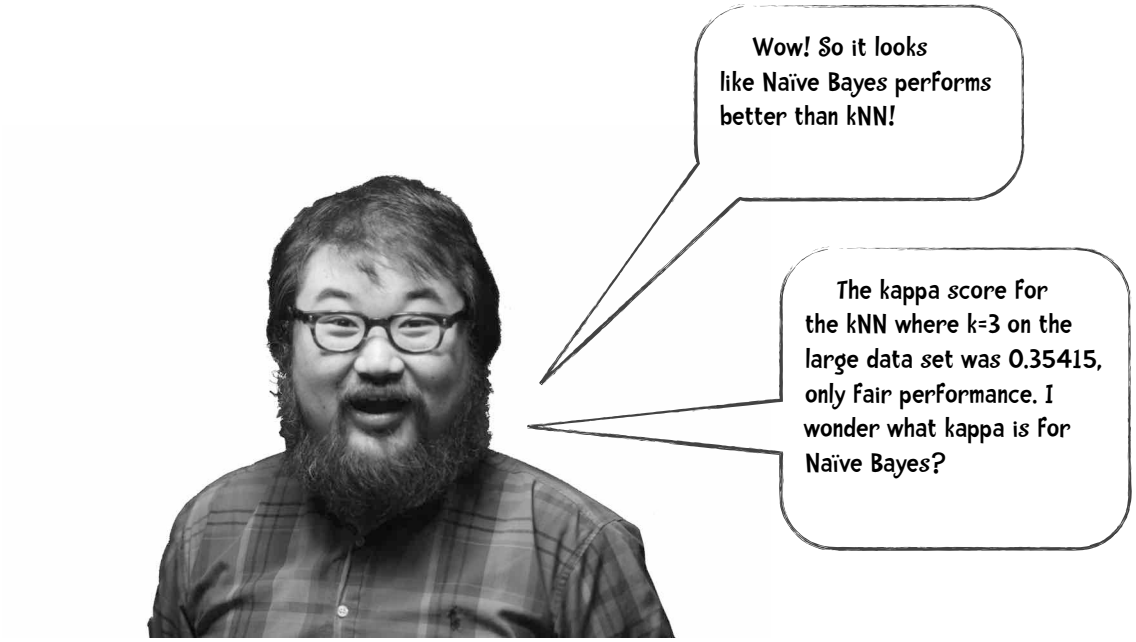
Is this any better than the Nearest Neighbor Algorithm?

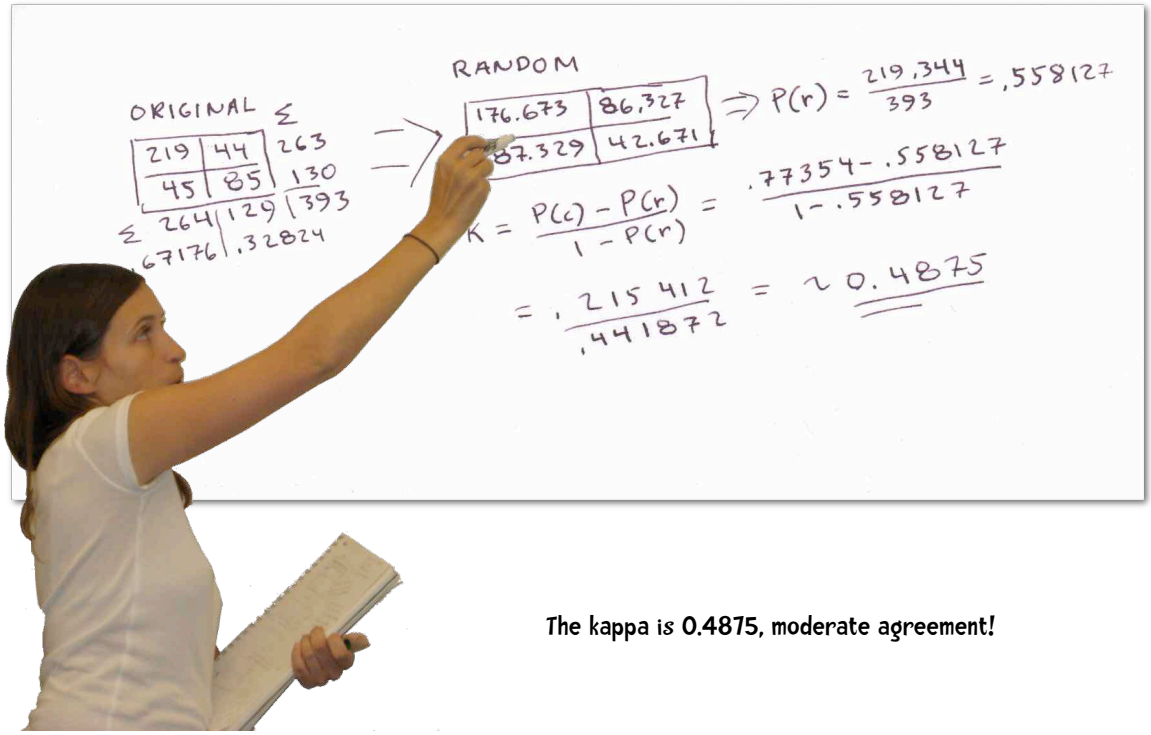
In Chapter 5 we evaluated how well the k Nearest Neighbor algorithm did with both the total Pima data set and a subset. Here are those results:

	pimaSmall	pima
k=1	59.00%	71.247%
k=3	61.00%	72.519%

Here are the results when we use Naïve Bayes with these two data sets:

	pimaSmall	pima
Bayes	72.000%	77.354%





The kappa is 0.4875, moderate agreement!

So for this example, Naïve Bayes is better than k Nearest Neighbors

Advantages of Bayes

- simple to implement (just counting things)
- need less training data than many other methods
- a good method to use if you want something that performs well and has good performance times.

Main disadvantage of Bayes:

It cannot learn interactions among features. For example, it cannot learn that I like foods with cheese and I like foods with rice but I do not like foods with both

Advantages of kNN

- simple to implement.
- does not assume the data has any particular structure—a good thing!
- large amount of memory needed to store the training set.

kNN

k Nearest Neighbors is a reasonable choice when the training set is large. kNN is extremely versatile and used in a large number of fields including recommendation systems, proteomics (the study of the entire protein set of an organism), the interaction among proteins, and image classification.

What enables us to multiple probabilities together is the fact that the events these probabilities represent are independent. For example, consider a game where we flip a coin and roll a die. These events are independent meaning what we roll on the die does not depend on whether we flip a heads or tails on the coin. And, as I just said, if events are independent we can determine their joint probability (the probability that they both occurred) by multiplying the individual probabilities together. So the probability of getting a heads and rolling a 6 is

$$P(heads \wedge 6) = P(heads) \times P(6) = 0.5 \times \frac{1}{6} = 0.08333$$

Let's say I alter a deck of cards keeping all the black cards (26 of them) but only retaining the face cards for the red suits (6 of them). That makes a 32 card deck. What is the probability of selecting a face card?

$$P(facecard) = \frac{12}{32} = 0.375$$

The probability of selecting a red card is

$$P(red) = \frac{6}{32} = 0.1875$$

What is the probability of selecting a single card that is both red and a face card? Here we do not multiply probabilities. We **do not** do

$$P(red \wedge facecard) = P(red) \times P(facecard) = 0.375 \times 0.185 = 0.0703$$

Here is what our common sense tells us. The chance of picking a red card is .1875. But if we pick a red card it is 100% likely it will be a face card. So it seems that the probability of picking a card that is both red and a face card is .1875.

Or we can start a different way. The probability of picking a face card is .375. The way the deck is arranged half the face cards are red. So the probability of picking a card that is both red and a face card is .375 * .5 = .1875.

Here we cannot multiply probabilities together because the attributes are not independent—if we pick red the probability of a face card changes—and vice versa.

In many if not most real world data mining problems there are attributes that are not independent.

Consider the athlete data. Here we had 2 attributes weight and height. Weight and height are not independent. The taller you get the more likely you will be heavier.

Suppose I have attributes zip code, income, and age. These are not independent. Certain zipcodes have big bucks houses others consist of trailer parks. Palo Alto zipcodes may be dominated by 20-somethings—Arizona zipcodes may be dominated by retirees.

Think about the music attributes—things like amount of distorted guitar (1-5 scale), amount of classical violin sound. Here many of these attributes are not independent. If I have a lot of distorted guitar sound, the probability of having a classical violin sound decreases.

Suppose I have a dataset consisting of blood test results. Many of these values are not independent. For example, there are multiple thyroid tests including free T4 and TSH. There is an inverse relationship between the values of these two tests.

Think about cases yourself. For example, consider attributes of cars. Are they independent? Attributes of a movie? Amazon purchases?

So, for Bayes to work we need to use attributes that are independent, but most real-world problems violate that condition. What we are going to do is just to assume that they are independent! We are using the magic wand of sweeping things under the rug™—and ignoreing this problem. We call it *naïve Bayes* because we are naïvely assuming independence even though we know it is not. It turns out that naïve Bayes works really, really, well even with this naïve assumption.

code it

Can you run the naïve Bayes code on our other data sets? For example, our kNN algorithm was 53% accurate on the auto MPG data set. Does a Bayes approach produce better results?

```
tenfold("mpgData/mpgData", "class attr  num  num  num  num  comment")
```

?????